Non-standard Queuing Policies: Definition of ARGESIM Benchmark C22

Peter Junglas[1*], Thorsten Pawletta[2]

[1]Dep. of Engineering "Dr. Jürgen Ulderup", PHWT Vechta/Diepholz, Schlesierstr. 13a, 49356 Diepholz, Germany; *peter@peter-junglas.de

[2]Wismar Univ. of Applied Sciences, Fac. of Engineering, Research Group CEA, PF 1210, 23952 Wismar, Germany

SNE 29(3), 2019, 111-115, DOI: 10.11128/sne.29.bn22.10481
Received: March 10, 2019; Revised: June 26, 2019;
Accepted: July 5, 2019
SNE - Simulation Notes Europe, ARGESIM Publisher Vienna,
ISSN Print 2305-9974, Online 2306-0271, www.sne-journal.org

Abstract. The ARGESIM benchmark C22 'Non-standard Queuing Policies' studies three non-standard queues that provide different ways to access entities inside a queue like detaching elements or reorder them: The reneging queue, where entities leave a queue after a given waiting time, the jockeying queue, where entities can switch to another shorter queue, and the classing queue, where at certain times entities with a given attribute ("class") are called to the front of the queue. A special focus lies on the management of concurrent events. The benchmark is especially suited for beginners in the field of discrete event modeling.

Introduction

In many applications of discrete event system simulation the modeling of queueing systems is of paramount importance. The basic paradigm describes abstract entities (customers, products, messages) that enter a queue, wait, until the corresponding server is free, and leave the queue, when they are selected for processing according to the queue discipline. Modifying the properties of server processes – such as service time distribution or the possibility of failure – and of the corresponding queues – e.g. their size or discipline – can change the overall system behaviour drastically. Furthermore one is often interested in specific statistical properties of the servers, queues and entities like mean uptime, queue length or average waiting time.

For this reason many simulation environments provide ready-to-use components of standard queues and servers, either as different blocks like in SimEvents [1] or combined in one `Process` module as in Arena [2]. Using parameters one can easily define the size of a queue, choose among a set of predefined disciplines (e.g. FIFO, LIFO, priority) or change the service time distribution.

But there are a lot of important examples, where the behaviour of the queue selection process or of the entities in the queue is more complex [3]. In this benchmark we will concentrate on the following three scenarios:

- *Jockeying:* the last entity in one of a set of FIFO queues can switch to another shorter queue.
- *Reneging:* entities wait in the queue only for a fixed maximal time and leave the queue and the system, if they are not served before.
- *Classing:* entities have a class attribute, similar to a priority, and advance to the front of the queue, when an external operator calls for their class number.

In a standard FIFO queue, an entity can only leave when it reaches the front of the queue. In contrast the three examples introduce additional ways to access entities inside a queue: detach the last element (jockeying), detach any element (reneging) or reorder all elements (classing). Frequently the standard queue components defined in simulation environments do not offer such access. This can make the implementation of a non-standard queue rather complicated, introducing a lot of additional internal events [4].

All three examples are variations of a simple basic queueing system containing four standard FIFO queues and servers. Its implementation should be straightforward, but it can be useful nevertheless, in order to scrutinize the exact system behaviour in the case of concurrent events. In the context of mainly stochastical processes this seems to be an unlikely case, but it is important e. g. to control the exact order of event cascades [5].

Therefore the models in this benchmark have to be implemented in two versions: A smaller deterministic model allows for exact comparisons and outputs plots of its dynamic behaviour, while a larger stochastic model produces statistical data of some relevant system variables.

1 Basic Queuing System

The simple queueing system shown in Figure 1 forms the basis of all examples. Its main purpose is to define all the details that are identical for the following special cases, but it also allows for some interesting variations that will be studied in the benchmark.

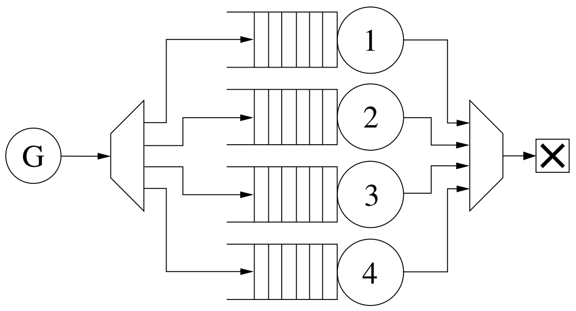

Figure 1: Basic queueing system with four queues.

The system contains a generator that creates a given number n_E of entities with fixed or stochastic interarrival times t_A, the first one starting always at $t = 1$. The entities have id attributes $1 \ldots n_E$ that are given in order of creation. They enter the system of $n_Q = 4$ numbered queues and servers, choosing the shortest line, including the server allocation. In case of several queues with minimal length the one with the smallest number is chosen. The queues have a FIFO (“First In First Out”) discipline and a potentially infinite capacity. Each server has a capacity of one and a fixed or stochastic service time t_S.

After being served the entities leave the system, e. g. they are terminated. The simulation stops, when all n_E entities have been terminated.

The *deterministic version* has $n_E = 100$ entities, constant interarrival time $t_A = 1$ and constant service time $t_S = 4.5$. Its simulation should produce two plots, one showing $ids(t)$, i.e. the ids of the last 20 outgoing entities over their termination time, preferably as stem or bar plot, the other the total queue length $l_{qt}(t)$ (i.e. the sum of the four queue lengths, without the server allocation) over the complete simulation time.

The *stochastic version* uses $n_E = 500$ entities, the interarrival times are exponentially distributed with a mean value of $t_A = 1$. The service times are computed using a symmetric triangular distribution with the most likely value $t_S = 4.5$ and a half-width $\Delta t_S = 2$, i. e. the possible values range from $t_{S,min} = 2.5$ to $t_{S,max} = 6.5$. The simulation should output the maximal and the average value of the total queue length $l_{qt}(t)$ (again defined as the sum of the four queue lengths, without the server allocation), where the average is defined as time average over the complete simulation time. Additional results are the maximum and average of the queue waiting times $t_{q,i}$ of entity i, where the service time is not included and the average is taken over all entities.

These two base models are pretty much standard, their implementation should present no difficulties. The two versions should be basically identical, especially no optimisations are allowed that depend on the special parameters of the deterministic input or server process.

The above description is not complete, insofar as the concrete order of concurrent events is not specified. This is an important issue at least for the deterministic version, where the results actually depend on such fine details. Therefore the concurrency order is generally specified in the following way:

1. an entity leaves a server,
2. a queued entity enters a server,
3. a new entity enters the system and chooses a queue.

For the variants described in the following sections additional event types may occur and the concrete order will be fixed accordingly. To study the methods, how the order of concurrent events can be fixed in a simulation environment, this benchmark includes another, optional model, namely a variant of the deterministic basic model with *another concurrency order*:

1. a new entity enters the system and chooses a queue,
2. an entity leaves a server,
3. a queued entity enters a server.

Another interesting problem is, how a simulation program deals with *large systems*. Especially for standard graphical environments the creation of a lot of queues and servers with copy and paste is a nuisance, there should be better ways to cope with the size. Therefore the benchmark includes another optional model, a variant of the stochastic version with 40 queues and servers. All parameters are as before except for the following:

$$n_Q = 40, \quad t_A = 0.1, \quad n_E = 5000.$$

2 Jockeying Queues

In a system with several queues *jockeying* means the process that an entity moves from one queue to another, usually shorter queue [3]. In the context of this benchmark it is specified in detail as follows:

- Jockeying happens immediately, when a queue (incl. server) is at least shorter by 2 than another one.
- In this case the last entity of the longer (*source*) queue leaves its queue and becomes the last entity of the shorter (*destination*) queue.
- If there are several destination queues, the one with the smallest queue number is chosen.
- In case of several possible source queues, the one is chosen that is nearest to the destination queue, i.e. where the absolute value of the difference of their queue numbers is minimal. If there are two such source queues (one on each side of the destination queue), the one with the smallest number is selected.

To complete the specification the order of concurrent events is given as:

1. an entity leaves a server,
2. a queued entity enters a server,
3. a jockeying entity changes its queue,
4. a new entity enters the system and chooses a queue.

Both variants of the basic model have to be augmented with the described jockeying behaviour. All parameters remain the same, as well as the standard output graphs and statistical values. For a jockeying entity the queue waiting time is defined as the sum of its waiting times in all queues it has visited (possibly many). Additionally the stochastic model should output the total number of jockeying events, while the output of the deterministic version should include a table showing the time of each jockeying event, the id of the jockeying entity and the numbers of the source and the destination queue. For conciseness, only the first five and the last five rows of the table have to be included in a benchmark report.

3 Reneging Queues

In this model entities that have entered a queue can leave it, before they are being served. This behaviour is called *reneging* [3]. There are many possible strategies, when entities renege, but in the benchmark example it is simply done, when the maximal waiting time $t_R = 9$ is reached. The order of concurrent events is as in the standard case, with the additional requirement that entering a server takes precedence over reneging, i. e. the order is:

1. an entity leaves a server,
2. a queued entity enters a server,
3. a queued entity reneges,
4. a new entity enters the system and chooses a queue.

Again both versions of the basic queueing system have to be extended to include the reneging of entities, using identical parameters and output graphs resp. values. For the computation of the average queue waiting time the time of the reneging entities – being t_R of course – is included. In addition the deterministic model should output a table of time and id of all reneging entities, while the stochastic model simply shows their total number.

4 Classing Queues

The last example system, called *classing queues*, is the most complex. It is inspired by a typical situation during the boarding of a plane: An operator calls “all passengers with seat numbers 15 – 30” to the front of the queue.

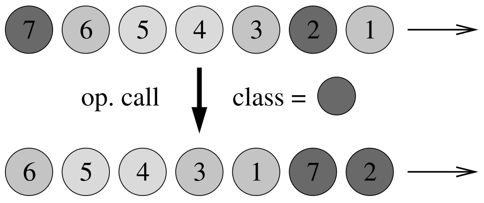

Figure 2: Result of an operator call (class $\triangleq$ color).

To mimic this each entity i is supplied with a fixed integer class c_i, where $1 \leq c_i \leq n_C$, with the number of classes given as $n_C = 5$. At certain times an operator calls for a class number, whereupon all entities with this class procede to the front of their queues. The relative order of the entities within this class remains intact, as does the order of the other entities among themselves (cf. Figure 2). Such a behaviour is similar to the standard priority queue, with the essential difference, that the meaning of "high priority" changes at runtime.

As before the classing behaviour has to be included in both variants of the basic queueing system. The way entities are assigned their class is different for the variant models: In the deterministic case the classes c_i are dealt in a round robin way in ascending order, starting with 1. In the stochastic version they are chosen randomly with equal probability $1/n_c$ for each class. An entity can only proceed to the server, if its class is the currently called class.

Unlike the boarding example, which ends after the boarding process is completed, the here defined "classing queue" should possibly run forever. Therefore the exact behaviour of the operator is defined as follows:

- Initially it waits for a fixed time $t_C = 10$, during which all incoming entities remain in the queue.
- After that it calls the classes in descending round-robin order, starting with the highest one ($n_C = 5$). This defines the current class identically for all queues, i.e. there is only one operator for the whole system.
- After a call it waits, until all entities of the current class have been served, before it calls the next class.
- If there are no entities in the system, the operator is stalled, until a new entity arrives.

The call of the operator has lowest priority among concurrent events, apart from that the order is as in the standard case:

1. An entity leaves a server,
2. a queued entity enters a server,
3. a new entity enters the system and chooses a queue,
4. the operator calls a new class and the queue is reordered accordingly.

The standard output graphs resp. statistical values have to be supplied, together with a table of the average and maximal values of the queue waiting times $t_{q,i}$ for each class, in both variants.

5 Specification of all Benchmark Tasks

All benchmark models have been described above with their exact parameters, together with the requested outputs of corresponding simulation runs. For easier reference, this section summarizes all items that a benchmark report should contain.

Basic Queuing System

1.1 A short description of the relevant parts of the basic model, using plots of the component structure, code snippets or whatever may be appropriate to understand the basic idea of the implementation.

1.2 plots of $ids(t)$ and $l_{qt}(t)$ (deterministic model),

1.3 results for max. and avg. of $l_{qt}(t)$ and $t_{q,i}$ (stochastic model),

1.4 (optional) a comparison of the implementations of the two concurrency variants together with a plot of $ids(t)$ for the variant model,

1.5 (optional) a comparison of the implementations of the standard and large stochastic models together with max. and avg. of $l_{qt}(t)$ and $t_{q,i}$ for the large model.

Jockeying Queues

2.1 A short description of the implementation of the jockeying queues,

2.2 plots of $ids(t)$ and $l_{qt}(t)$ and a table displaying the first five and the last five jockey events (deterministic model),

2.3 results for max. and avg. of $l_{qt}(t)$ and $t_{q,i}$ and the number of jockey events (stochastic model).

Reneging Queues

3.1 A short description of the implementation of the reneging queues,

3.2 plots of $ids(t)$ and $l_{qt}(t)$ and a table displaying the reneging events (deterministic model),

3.3 results for max. and avg. of $l_{qt}(t)$ and $t_{q,i}$ and the number of reneging entities (stochastic model).

Classing Queues

4.1 A short description of the implementation of the classing queues,

4.2 plots of $ids(t)$ and $l_{qt}(t)$ and a table displaying class statistics (deterministic model),

4.3 results for max. and avg. of $l_{qt}(t)$ and $t_{q,i}$ and a table displaying class statistics (stochastic model).

Solutions should be accompanied by the complete source code of all models to make them accessible on the ARGESIM Benchmark server.

6 Conclusion

Depending on the simulation environment used, some of the tasks can be very easy or may require tricky modeling and implementation ideas. Probably a special difficulty will be to guarantee the specified order of concurrent events. The production of plots and statistical results tests the corresponding capabilities of the simulation environment, but can of course be done using data export and external programs.

Since all system examples are rather small and don't need a special mathematical or modeling background, the benchmark is suited for beginners in the field of modeling and simulation.

Acknowledgement

The first author (P. J.) likes to thank the team of the CEA research group in Wismar for the warm hospitality extended to him and many helpful discussions.

References

[1] Li W, Mani R, Mosterman P. Extensible discrete-event simulation framework in SimEvents. *Proc. 2016 Winter Simulation Conference*; 2016 Dec; Arlington. New Jersey: IEEE. 943-954.

[2] Kelton W, Sadowski R, Zupick N. *Simulation with Arena*. 6th ed. New York: McGraw-Hill; 2015. 656 p.

[3] Hassin R, Haviv M. *To queue or not to queue: Equilibrium behavior in queueing systems*. Boston: Kluwer Academic Publishers; 2003. 191 p.

[4] Austermann L, Junglas P, Schmidt J, Tiekmann C. Conceptional problems of transaction-based modeling and its implementation in SimEvents 4.4. *Simulation Notes Europe SNE*. 2017; 27(3): 137–142. doi: 10.11128/sne.27.tn.10383

[5] Junglas P. Pitfalls using discrete event blocks in Simulink and Modelica. In: *Proc. of ASIM-Workshop STS/GMMS*; 2016 Mar; Lippstadt. 90-97. ISBN 978-3-901608-48-3.